1 Introduction

This draft presents an introduction to CPU syncronization algorithms with empasys on *mutual exclusion* problems.

The problem of *mutual exclusion* can be stated as: given at least two concurrent process, sharing a resource, one must must provide for the mutual exclusion access to the resource: at most one process is accessing the resource at any given time. There are many solutions, each one tries to guarantee, at least:

- *mutual exclusion*;
- *deadlock* avoidance;
- *liveness*.

Any solution must be valid under any interleaving of steps.

2 Dekker's solution

In this section, we use *Pomella*, the language of *spin*, to model a version of the Dekker's solution for the mutual exclusion problem, when two processes are involved. To understand the solutions is important to remember that the semantics of *Pomella* assumes:

- each statement is atomic;
- *send/receive* are blocking;
- any boolean statement until it is true;
- the *do–od* and *if–fi* commands are nondeterministic;
- if the only statement for a process to execute is blocked then the whole process blocks.

2.1 A busy wait *solution*

The first soultion is borrowed from [?]. The *do–od* in the *user* process implements a busy wait. Extract the code from this document and run it under *xspin* to discover that this solution does not ensure *progress* under any interleaving. Why? Is the version of the Dekker's solution presented by [?] incorrect?

```
#define true 1
#define false 0
bool in[2];
bool c[2];
int turn;


proctype user(int i)
{
   int next;

   next  = (i+1) % 2;
   c[i]=true;
   turn = next;

   do
     :: c[next] && turn != i ->
accept: skip;
     :: !c[next] || turn==i ->
progress: break;
   od;

   in[i]=true;
   assert(in[next]==false);
   in[i]=false;
   c[i]=false;
}



init
```

```
{
  c[0]=false;c[1]=false;
  turn=0;
  in[0]=false;
  in[1]=false;
  atomic{
    run user(0);
    run user(1);
  }
}
```

2.2 A solution with blocking

In this section, we avoid the problems with interleaving by forcing one process to yield the processor when it finds that it cannot enter its critical section. The statement

(!c[next] —— turn == i);

suspends process i until one *C[next]* or *(turn==i)* is true.

```
#define true 1
#define false 0
bool in[2];
bool c[2];
int turn;


proctype user(int i)
{
   int next;

   next  = (i+1)%2;
   c[i]=true;
   turn = next;
   (!c[next] || turn == i);
progress:
```

```
    in[i]=true;
    assert(in[next]==false);
    in[i]=false;
    c[i]=false;
}



init
{
  c[0]=false;c[1]=false;
  turn=0;
  in[0]=false;
  in[1]=false;
  atomic{
    run user(0);
    run user(1);
  }
}
```

3 The *bakey* algorithm

The bakery algorithm was introduced by *Leslie Lamport* in the 1974, and is certainly a classic in Computer Science. It is a solution for the mutual exclusion problem that contemplates the case when more than one process compete for the same resource.

A constant *NP* is introduced to define the number of processes.

Each process i uses two variables:

- *chossing[i]* that is used to mark the beginning and ending of the ticket acquisition phase: region delimited in *try(i)*.
- *number[i]* to hold a ticket number.

```
#define true 1
#define false 0
#define NP 3

bool choosing[NP];
byte number[NP];
byte count;

```

Process i chooses a ticket number in *try(i)*, and it is the maximum ticket of all processes incremented by one.

```
inline try(i){
  choosing[i]=true;
  int j,mx;
  mx=number[0];
  j=0;
  do
   :: j==NP -> break;
   :: j<NP && mx<number[j] -> mx=number[j];j=j+1;
   :: else -> j=j+1;
  od;
  number[i]=mx+1;
  choosing[i]=false
}
```

Given a ticket *number[i]*, process i passes *door(i)* and enters the critical section if it it has a better ticket number than any other competitor. A ticket cannot be compared if its owner is still acquiring it, *try* region.

```
inline door(i){
  int j;
  j=0;
  do
   :: j<NP -> (!choosing[j]);
              (number[j] == 0 ||
```

```
               number[i]<number[j] ||
               (number[i]==number[j] && i < j));
            j=j+1;
   :: j==i -> j=j+1;
   :: j==NP -> break;
  od
}
```

A process leaves the critical section by returning its ticket number to zero.

```
inline exit(i){
  number[i]=0
}
```

The *proctype user* models processes that would use the bakery algorithm to achieve mutual exclusion. The variable *count* is used to assert that the number of processes in the critical section is less than 2.

```
proctype user(byte i)
{
   try(i);
   door(i);

progress:
   count=count+1;
   skip;
   count=count-1;
   exit(i)
}

```

Process *monitor* implements an invariant for the system asserting that either 0 or 1 process is in the critical section.

```
proctype monitor(){
  assert(count == 0 | count == 1);
}
```

Process *init* starts *NP* copies of *user* and one copy of *monitor*.

```
init{
  count = 0;
  atomic{
    int j;
    j=0;
    do
     :: j<NP -> run user(j); j=j+1;
     :: else break;
    od;
    run monitor();
  }
}

```

4 The *Joint Smokers* problem

Bill, Osama and Yasser would like to forget old differences, and smoke the pot of peace. The problem is that each one of them needs pot, wrapper, and matches, and if they don't get them from time to time, they may start plotting new wars in the name of God and freedom. Now, Bill has got the pot (of course), Osama will always have the matches, and poor Yasser has bad quality wapper. There come George, Tony, and Ariel and they seem to have the solution for this much unbecoming situation. George produces pot and matches (and guns, in case you need them); Tony supplies matches and wrapper (well... and guns; you would never know when you need them with all these arabs and latin americans around); Ariel supplies wapper and pot (and, of course, he is great at burning guns and bloody arab meat. And jewish meat as well. Why not? What if you must burn a few thousand of your own just to ensure that even more of the *other kind* get killed?) Can you help Bill, Osama and Yasser to get supplies from the three God's angels to start getting comfortably numb? Use a strong semaphores in your solution. Can you improve this game to find a part for Fernando, where he does not play the joker? And if Osama and George decide to renew old family ties and restart their old sacred association for selling guns and oil? Can you find them a market before they burn us all?